Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа № 8

Тема: «Программа, управляемая событиями»

Выполнил: студент группы РИС-22-2б

Мизёв В.А. ф

Проверил: доцент кафедры ИТАС

Полякова О.А.

г. Пермь – 2024

*Постановка задачи:*

1. Определить иерархию пользовательских классов. Во главе иерархии должен стоять абстрактный класс с чисто виртуальными методами для ввода и вывода информации об атрибутах объектов.
2. Реализовать конструкторы, деструктор, операцию присваивания, селекторы и мдификаторы.
3. Определить класс-группу на основе структуры, указанной в варианте.
4. Для групп реализовать конструкторы, деструктор, методы для добавления и удаления элементов в группу, метод для просмотра группы, перегрузить операцию для получения информации о размере группы.
5. Определить класс Диалог – наследника группы, в котором реализовать методы для обработки событий.
6. Добавить методы для обработки событий группой и объектами пользовательских классов.
7. Написать тестирующую программу.
8. Нарисовать диаграмму классов и диаграмму объектов.

*Задание вариант 1:*

Базовый класс: ЧЕЛОВЕК (Person)

Имя – string

Возраст – int

Производный класс

Студент (Student)

Рейтинг – float

Группа – Вектор (Vector).

Команды:

- создать группу (формат команды: m количество элементов группы).

- добавить элемент в группу (формат команды: +)

- удалит элемент из группы (формат команды -)

- вывести информацию об элементах группы (формат команды: s)

- вывести информацию об элементе группы (формат команды :? Номер объекта в группе)

- конец работы (формат команды: q)

*Анализ задачи:*

Для класс-группы реализуем класс Dialog, который будет наследоваться от класса Vector. Также необходимо создать структуру Event, в которой будет хранится enum список команд, Type – тип сообщения и size\_t arg, для аргументов.

Класс Dialog будет состоять из следующих методов execute(), pollEvent(), handleEvent(), clearEvent(). Поля класса m\_state – для текущего состояния, m\_commands – строка с командами, где

+: добавить элемент,

-: удалить последний элемент,

s: показать весь массив,

n: показывает весь массив, выводя только имена,

l: выводит размер массива

q: выход из программы

Для объектов создадим абстрактный базовый класс object с методами show, input и handleEvent(). Класс Person наследует от класса Object, а класс Student будет наследоваться от класса Person. Для всех классов перегрузить оператор ввода, вывода и присваивания, а также реализовать соответствующие get’еры и set’еры для установки данных, таких как имя, возраст, рейтинг.

Класс вектор взять из прошлой лабораторной работы и дополнить следующими методами: pop\_back() – для удаления последнего элемента, show() – для вывода массива данных и handleEvent() – для обработки событий объектов.

*Код программы:*

Прикреплён в приложении 1.

*UML-диаграмма класса:*

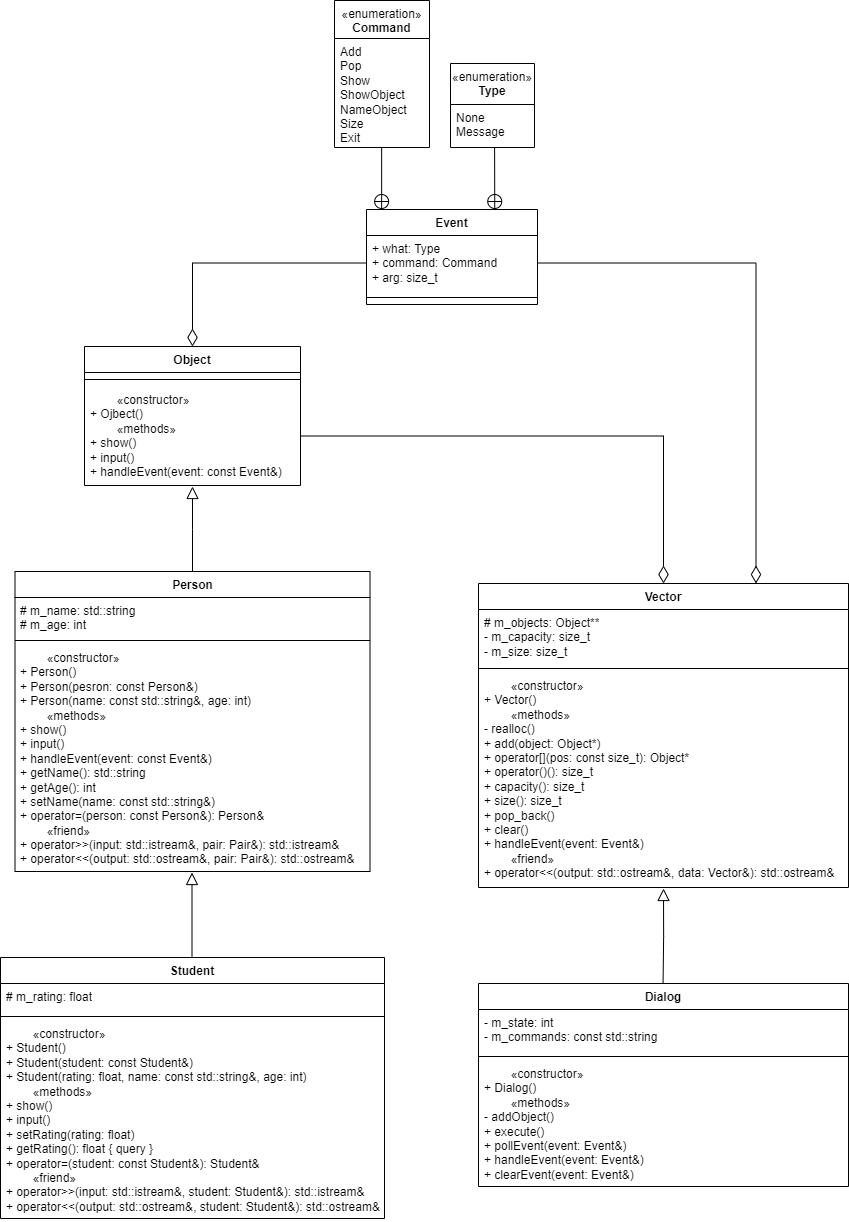


Рисунок 1 - UML-диаграмма классов

*Диаграмма объектов:*

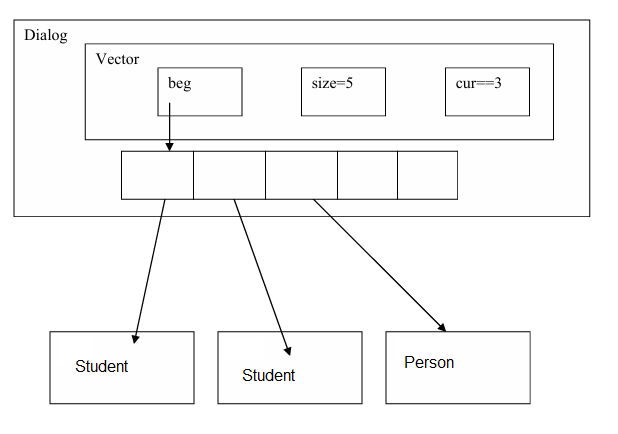


Рисунок 2 - диаграмма объектов

*Скриншот работы программы:*

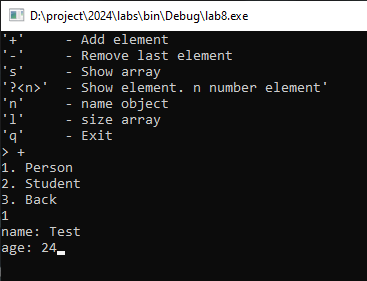


Рисунок 3 – Выполнение команды “+”

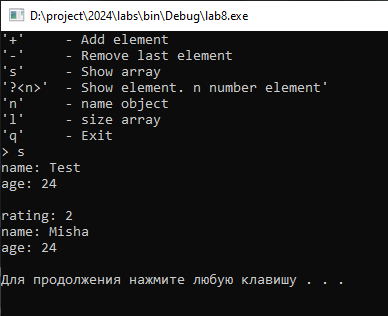


Рисунок 4 - Выполнение команды “s”

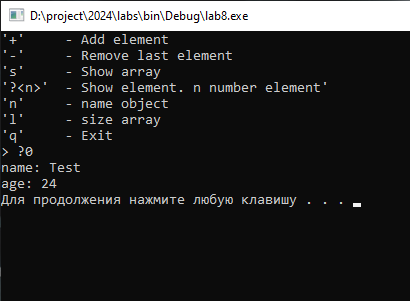


Рисунок 1 - Выполнение команды “?”

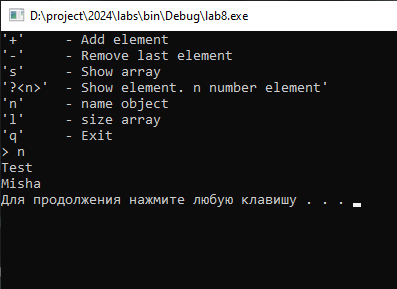


Рисунок 2 - Выполнение команды “n”

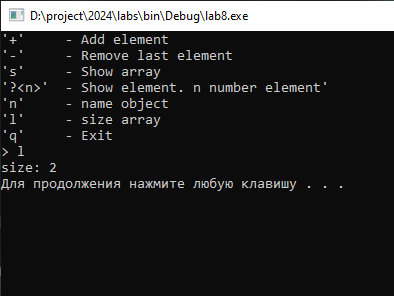


Рисунок 3 - Выполнение команды “l”

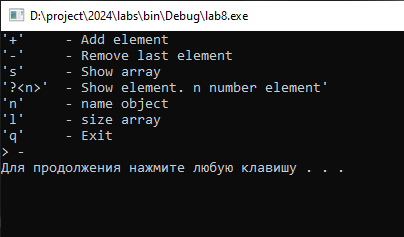


Рисунок 4 - Выполнение команды “-“

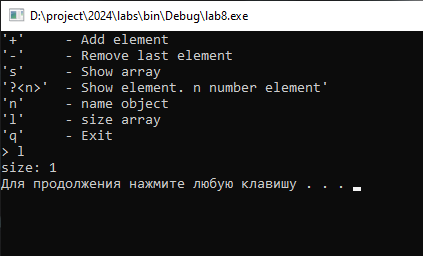


Рисунок 5 - Выполнение команды “l”

*Контрольные вопросы:*

1.Что такое класс-группа? Привести примеры таких классов.

Класс-группа (или класс) - это абстракция, которая объединяет объекты, обладающие общими чертами. Классы служат шаблонами для создания объектов и могут содержать как данные, так и методы. Примеры таких классов могут быть "Круг", "Прямоугольник", "Список", "Книга" и т.д.

2. Привести пример описания класса-группы Список (List).

class List {

private:

int \*arr; // Массив для хранения элементов

int size; // Размер списка

public:

List(int s) : size(s) { arr = new int[size]; } // Конструктор

// Дополнительные методы...

};

3. Привести пример конструктора (с параметром, без параметров, копирования) для

класса-группы Список.

class List {

private:

int \*arr;

int size;

public:

List() : size(0), arr(nullptr) {} // Конструктор без параметров

List(int s) : size(s) { arr = new int[size]; } // Конструктор с параметром

List(const List &other) : size(other.size) { // Конструктор копирования

arr = new int[size];

for (int i = 0; i < size; ++i) {

arr[i] = other.arr[i];

}

}

// Деструктор и другие методы...

};

4. Привести пример деструктора для класса-группы Список.

class List {

private:

int \*arr;

int size;

public:

~List() { delete[] arr; } // Деструктор

// Другие методы...

};

5. Привести пример метода для просмотра элементов для класса-группы Список.

class List {

// ... (предыдущий код)

public:

void printElements() {

for (int i = 0; i < size; ++i) {

std::cout << arr[i] << " ";

}

std::cout << std::endl;

}

};

6. Какой вид иерархии дает группа?

Группа может предоставить иерархию классов, в которой классы могут наследовать свойства и методы друг от друга. Это может быть иерархия через множественное наследование или простое наследование, где базовый класс предоставляет общий функционал для производных.

7. Почему во главе иерархии классов, содержащихся в группе объектов должен

находится абстрактный класс?

Абстрактный класс является шаблоном для других классов и определяет интерфейс, который все производные классы должны реализовать. Это позволяет реализовать полиморфизм и обеспечить поддержку общего поведения для производных классов.

8. Что такое событие? Для чего используются события?

Событие - это сигнал о том, что произошло некое действие (например, нажатие кнопки, изменение данных и т.д.). События используются для обработки асинхронных действий и взаимодействия между объектами, позволяя отделить логику обработки от источника события.

9. Какие характеристики должно иметь событие-сообщение?

Событие-сообщение должно содержать:

- Тип события (что именно произошло).

- Данные, связанные с событием (например, координаты мыши, нажатая клавиша).

- Время, когда произошло событие (если это важно для контекста).

- Источник события (объект, который вызвал событие).

10. Привести пример структуры, описывающей событие.

struct Event {

enum Command {

Add = 0,

Pop,

Show,

ShowObject,

NameObject,

Size,

Exit

};

enum Type {

None = 0,

Message

};

Type what;

Command command;

size\_t arg;

};

Где Type – тип сообщения, Command – команда, которую нужно выполнить, size\_t arg – аргументы (для хорошего варианта, лучше использовать указатель на void\*)

11. Задана структура события.

struct TEvent {

int what;

union {

MouseEventType mouse;

KeyDownEvent keyDown;

MessageEvent message;

}

}

Какие значения, и в каких случаях присваиваются полю what?

В случаи событий нажатия на кнопки мыши в MouseEventType записывается тип события нажатие кнопки или отжатие, в keyDown код нажатой кнопки, а в MessageEvent – сообщение.

12. Задана структура события

struct TEvent {

int what; //тип события

union {

int command; //код команды

struct { //параметры команды

int message;

int a;

}

}

};

Какие значения, и в каких случаях присваиваются полю command?

Присваивается код команды, которую необходимо выполнить при появления

данного события.

13. Задана структура события.

struct TEvent {

int what; //тип события

union {

int command; //код команды

struct { //параметры команды

int message;

int a;

}

}

};

Для чего используются поля a и message?

В данной структуре TEvent, поля a и message используются для передачи дополнительных параметров в контексте конкретного события.

- поле message: Это поле может использоваться для передачи информации о сообщении, которое связано с событием. Например, если событие связано с пользовательским вводом, message может содержать текст или код сообщения, который указывает на конкретное действие пользователя или результат обработки.

- поле a: Это поле может служить для передачи специфических параметров, связанных с событием. Например, оно может использоваться для передачи дополнительной информации, которая требуется для выполнения команды или реакции на событие. Это может быть код, значение, состояние или любой другой параметр, который может быть важен для обработки данного события.

14. Какие методы необходимы для организации обработки сообщений?

Для обработки сообщений обычно необходимы методы для добавления сообщений в очередь, их обработки и удаления. Эти методы могут включать AddMessage, ProcessMessages и ClearMessages.

class MessageQueue {

public:

void AddMessage(const std::string& msg) {

queue.push\_back(msg);

}

void ProcessMessages() {

for (const auto& msg : queue) {

std::cout << "Processing message: " << msg << std::endl;

}

queue.clear();

}

private:

std::vector<std::string> queue;

};

15. Какой вид имеет главный цикл обработки событий-сообщений?

Главный цикл обработки событий-сообщений обычно включает в себя ожидание новых сообщений и их последующую обработку.

16. Какую функцию выполняет метод ClearEvent()? Каким образом?

Метод ClearEvent() обычно очищает или сбрасывает состояние события, чтобы оно не обрабатывалось повторно.

17. Какую функцию выполняет метод HandleEvent ()?Каким образом?

Метод HandleEvent() обрабатывает конкретное событие. Обычно он проверяет состояние события и выполняет соответствующие действия.

18. Какую функцию выполняет метод GetEvent ()?

Метод GetEvent() обычно возвращает текущее событие или сообщение для обработки.

19. Для чего используется поле End State? Какой класс (объект) содержит это поле?

Поле End State используется для обозначения завершающего состояния объекта или системы. Обычно оно содержится в классах, управляющих состояниями, как, например, в автоматах состояний.

20. Для чего используется функция Valid()?

Функция Valid() используется для проверки корректности данных.

**ПРИЛОЖЕНИЕ 1**

lab8.cpp

//-------------------------------------

#include <iostream>

#include "lab8\_dialog.hpp"

//-------------------------------------

#define IDENT\_PRINT printf("\n--------------------------\n\n")

//-------------------------------------

int main() {

Dialog dialog;

return dialog.execute();

}

//-------------------------------------

lab8\_dialog.hpp

//-------------------------------------

#ifndef DIALOG\_HPP\_INCLUDED

#define DIALOG\_HPP\_INCLUDED

//-------------------------------------

#include "lab8\_vector.hpp"

//-------------------------------------

struct Event {

enum Command {

Add = 0,

Pop,

Show,

ShowObject,

NameObject,

Size,

Exit

};

enum Type {

None = 0,

Message

};

Type what;

Command command;

size\_t arg;

};

//-------------------------------------

class Dialog : public Vector {

public:

Dialog();

~Dialog();

int execute();

void pollEvent(Event& event);

void handleEvent(Event& event);

void clearEvent(Event& event);

private:

int m\_state;

const std::string m\_commands = "+-s?nlq";

void addObject();

};

//-------------------------------------

#endif // DIALOG\_HPP\_INCLUDED

//-------------------------------------

lab8\_dialog.cpp

//-------------------------------------

#include "lab8\_dialog.hpp"

#include "lab8\_object.hpp"

#include <windows.h>

//-------------------------------------

Dialog::Dialog() {

m\_state = 1;

}

//-------------------------------------

Dialog::~Dialog() {

//

}

//-------------------------------------

void Dialog::addObject() {

uint32\_t value;

std::cout << "1. Person\n2. Student\n3. Back\n";

std::cin >> value;

if(value > 3)

throw std::runtime\_error("error type not found!");

else if(value == 3)

return;

value--;

Object\* object = value ? new Student() : new Person;

object->input();

add(object);

}

//-------------------------------------

int Dialog::execute() {

Event event;

while(m\_state) {

pollEvent(event);

handleEvent(event);

}

return m\_state;

}

//-------------------------------------

void Dialog::pollEvent(Event& event) {

system("cls");

std::string command;

char code;

std::cout << "'+' - Add element\n"

<< "'-' - Remove last element\n"

<< "'s' - Show array\n"

<< "'?<n>' - Show element. n number element'\n"

<< "'n' - name object\n"

<< "'l' - size array\n"

<< "'q' - Exit\n";

std::cout << "> ";

std::cin >> command;

if(!command.empty())

code = command[0];

if(m\_commands.find(code) >= 0) {

event.what = Event::Type::Message;

switch(code) {

case '+': event.command = Event::Add; break;

case '-': event.command = Event::Pop; break;

case 's': event.command = Event::Show; break;

case '?': event.command = Event::ShowObject; break;

case 'n': event.command = Event::NameObject; break;

case 'l': event.command = Event::Size; break;

case 'q': event.command = Event::Exit; break;

}

if(command.size() > 1)

event.arg = std::atoi(&command[1]);

return;

}

event.what = Event::Type::None;

}

//-------------------------------------

void Dialog::handleEvent(Event& event) {

if(event.what == Event::Type::Message) {

switch(event.command) {

case Event::Add: addObject(); clearEvent(event); break;

case Event::Pop: pop\_back(); clearEvent(event); break;

case Event::Show: show(); clearEvent(event); break;

case Event::ShowObject:

if(event.arg >= this->size()) {

std::cerr << "event.arg >= m\_size" << std::endl;

break;

}

(\*this)[event.arg]->show();

clearEvent(event);

break;

case Event::Size:

std::cout << "size: " << this->size() << std::endl;

break;

case Event::Exit:

m\_state = 0;

clearEvent(event);

break;

default:

Vector::handleEvent(event);

clearEvent(event);

break;

}

}

if(event.command != Event::Exit)

system("pause");

}

//-------------------------------------

void Dialog::clearEvent(Event& event) {

event.what = Event::Type::None;

}

//-------------------------------------

lab8\_object.hpp

//-------------------------------------

#ifndef LAB8\_OBJECT\_H\_INCLUDED

#define LAB8\_OBJECT\_H\_INCLUDED

//-------------------------------------

#include <iostream>

#include "lab8\_dialog.hpp"

//-------------------------------------

class Object {

public:

Object() = default;

virtual ~Object() = default;

virtual void show() = 0;

virtual void input() = 0;

virtual void handleEvent(const Event& event) = 0;

};

//-------------------------------------

class Person : public Object {

public:

Person();

Person(const Person& person);

Person(const std::string& name, int age);

~Person();

virtual void show() { std::cout << \*this << std::endl; }

virtual void input() { std::cin >> \*this; }

virtual void handleEvent(const Event& event);

const std::string& getName() const { return m\_name; }

int getAge() const { return m\_age; }

void setName(const std::string& name) { m\_name = name; }

void setAge(int age) { m\_age = age; }

Person& operator=(const Person& person);

friend std::istream& operator>>(std::istream& input, Person& person);

friend std::ostream& operator<<(std::ostream& output, Person& person);

protected:

std::string m\_name;

int m\_age;

};

//-------------------------------------

class Student : public Person {

public:

Student();

Student(const Student& student);

Student(float rating, const std::string& name, int age);

~Student();

virtual void show() { std::cout << \*this << std::endl; }

virtual void input() { std::cin >> \*this; }

void setRating(float rating) { m\_rating = rating; }

float getRating() const { return m\_rating; }

Student& operator=(const Student& student);

friend std::istream& operator>>(std::istream& input, Student& student);

friend std::ostream& operator<<(std::ostream& output, Student& student);

protected:

float m\_rating;

};

//-------------------------------------

#endif // LAB8\_OBJECT\_H\_INCLUDED

//-------------------------------------

lab8\_object.cpp

//-------------------------------------

#include "lab8\_object.hpp"

#include "lab8\_dialog.hpp"

//-------------------------------------

//-------------------------------------

// class Person

//-------------------------------------

//-------------------------------------

Person::Person() {

m\_age = 0;

}

//-------------------------------------

Person::Person(const Person& person) :

Person(person.m\_name, person.m\_age) {

//

}

//-------------------------------------

Person::Person(const std::string& name, int age) {

m\_name = name;

m\_age = age;

}

//-------------------------------------

Person::~Person() {

//

}

//-------------------------------------

void Person::handleEvent(const Event& event) {

if(event.what == Event::Type::Message) {

switch(event.command) {

case event.NameObject:

std::cout << m\_name << std::endl;

break;

default: break;

}

}

}

//-------------------------------------

Person& Person::operator=(const Person& person) {

if(this == &person)

return \*this;

m\_name = person.m\_name;

m\_age = person.m\_age;

return \*this;

}

std::istream& operator>>(std::istream& input, Person& person) {

std::cout << "name: ";

input >> person.m\_name;

std::cout << "age: ";

input >> person.m\_age;

return input;

}

std::ostream& operator<<(std::ostream& output, Person& person) {

return output << "name: " << person.m\_name

<< "\nage: " << person.m\_age;

}

//-------------------------------------

//-------------------------------------

// class Student

//-------------------------------------

//-------------------------------------

Student::Student() : Person() {

m\_rating = 0;

}

//-------------------------------------

Student::Student(const Student& student) :

Student(student.m\_rating, student.m\_name, student.m\_age) {

//

}

//-------------------------------------

Student::Student(float rating, const std::string& name, int age) :

Person(name, age) {

m\_rating = rating;

}

//-------------------------------------

Student::~Student() {

//

}

//-------------------------------------

Student& Student::operator=(const Student& student) {

if(this == &student)

return \*this;

\*((Person\*)this) = (Person)student;

m\_rating = student.m\_rating;

return \*this;

}

//-------------------------------------

std::istream& operator>>(std::istream& input, Student& student) {

std::cout << "rating: ";

input >> student.m\_rating;

input >> ((Person&)student);

return input;

}

//-------------------------------------

std::ostream& operator<<(std::ostream& output, Student& student) {

return output << "rating: " << student.m\_rating << "\n" << ((Person&)student);

}

//-------------------------------------

lab8\_vector.hpp

//-------------------------------------

#ifndef LAB8\_VECTOR\_HPP\_INCLUDED

#define LAB8\_VECTOR\_HPP\_INCLUDED

//-------------------------------------

#include <iostream>

#include <cstdlib>

#include <cstdint>

//-------------------------------------

#define BASE\_SIZE 4

//-------------------------------------

class Object;

class Event;

class Vector {

public:

Vector();

~Vector();

void add(Object\* object);

Object\* operator[](const size\_t pos) { return m\_size < pos ? nullptr : m\_objects[pos]; }

size\_t operator()() { return m\_size; }

size\_t capacity() const { return m\_capacity; }

size\_t size() const { return m\_size; }

void show() { std::cout << \*this << std::endl; }

void pop\_back();

void clear();

virtual void handleEvent(Event& event);

friend std::ostream& operator<<(std::ostream& output, Vector& data);

protected:

Object\*\* m\_objects;

private:

size\_t m\_capacity,

m\_size;

void realloc();

};

//-------------------------------------

#endif // LAB8\_VECTOR\_HPP\_INCLUDED

//-------------------------------------

lab8\_vector.cpp

//-------------------------------------

#include "lab8\_vector.hpp"

#include "lab8\_object.hpp"

//-------------------------------------

Vector::Vector() {

m\_size = 0;

m\_capacity = 0;

this->realloc();

}

//-------------------------------------

Vector::~Vector() {

this->clear();

}

//-------------------------------------

void Vector::realloc() {

if(m\_size < m\_capacity)

return;

if(m\_capacity == 0) {

m\_capacity = BASE\_SIZE;

m\_objects = new Object\*[m\_capacity];

return;

}

m\_capacity\*=2;

Object\*\* array = new Object\*[m\_capacity];

for(size\_t i = 0; i < m\_size; ++i)

array[i] = m\_objects[i];

delete []m\_objects;

m\_objects = array;

}

//-------------------------------------

void Vector::add(Object\* object) {

this->realloc();

m\_objects[m\_size] = object;

m\_size++;

}

//-------------------------------------

void Vector::pop\_back() {

if(m\_size == 0)

return;

Object\* obj = (\*this)[m\_size-1];

if(obj != nullptr)

delete obj;

obj = nullptr;

m\_size--;

}

//-------------------------------------

void Vector::clear() {

for(size\_t i = 0; i < m\_size; ++i) {

if(m\_objects[i] != nullptr)

delete m\_objects[i];

}

delete []m\_objects;

m\_size = 0;

m\_capacity = 0;

}

//-------------------------------------

void Vector::handleEvent(Event& event) {

if(event.what == Event::Type::Message) {

for(size\_t i = 0; i < m\_size; ++i)

m\_objects[i]->handleEvent(event);

}

}

//-------------------------------------

std::ostream& operator<<(std::ostream& output, Vector& data) {

for(size\_t i = 0; i < data.size(); ++i) {

data[i]->show();

if(i != data.size()-1)

std::cout << std::endl;

}

return output;

}

//-------------------------------------